Atelier Data Science

Deep learning practice 1

Neural Net Foundations

Irina Proskurina

Irina.Proskurina@univ-lyon2.fr

Laboratoire ERIC – Université Lyon 2

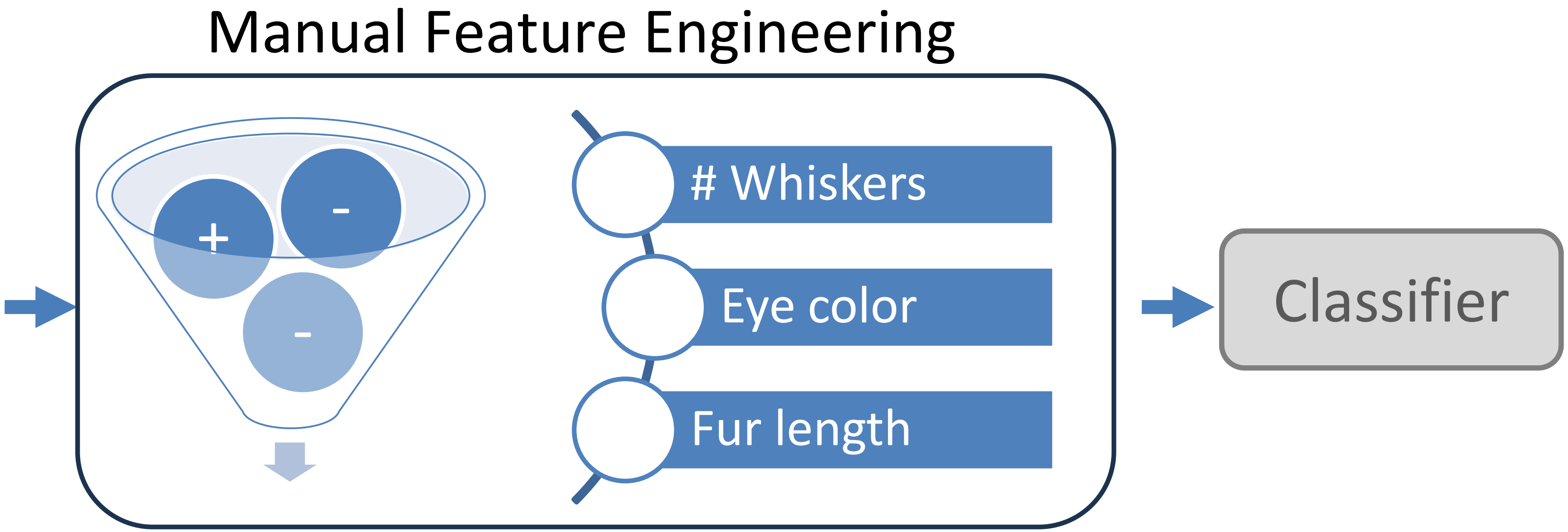
Course Plan

- Neural Net Foundations, Backpropagation
- CNNs
- Optimization in DL
- Batch Normalization
- Pre-trained Transformers
- Transformer Fine-tuning
- Diffusion Models
- GANs

Link : https://github.com/upunaprosk/ul2-atelier-data-science

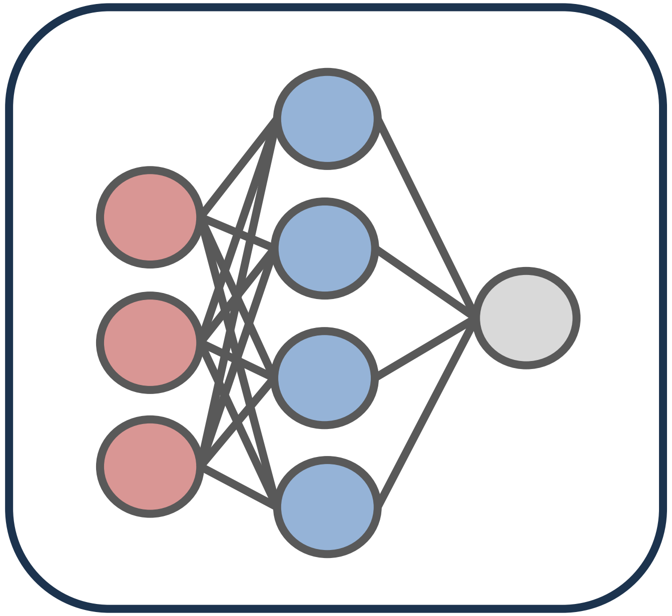
Traditional Computer Vision

1. Feature engineering (whiskers, the shape of the ears, the length of the tail, etc.)
2. Train gradient boosting on features



But Feature engineering is too difficult and costly!

Modern Deep Computer Vision



Neural Network

Automated Feature Extraction + Classification

http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf

Traditional NLP

1. Count the frequency with which a specific word appears following another ones
2. Generate the next word based on this distribution

"Manure, almond gelato and frozen pies, you are also had it was in one but it will post office buildings s ucks). their chinese food. comfort food while they liked their lids ripped off. it an early morning of jon still a spade so maybe too much. the same. but, at the baked rigatoni, and not in other options and it see ms odd taste). our visit). i go to nfl kickoff arrived with…"

Example of text generation using Markov Chains

https://medium.com/analytics-vidhya/making-a-text-generator-using-markov-chains-e17a67225d10

Modern NLP

1. GPT-3 — a neural network trained on a massive corpus of texts

> The article you are writing about is going to be based around this new technology, so you have been spending a lot of time playing around with it. You have also been using your own brain to test out the new models, which is something no one else in the world has done. As a result, you have become somewhat obsessed with it. You constantly think about how it can create such fantastic sentences and how it might be used to solve the world's problems.

Recent achievements

- Images and videos
- Three-dimensional computer vision
- Text generation
- Audio-to-Audio models
- Data generation

Useful Links

- https://www.deeplearningbook.org
- https://cs231n.github.io/convolutional-networks/
- https://course.fast.ai

Why do we need neural networks?

Because they can model non-linear complex relationships in data!

Example
Predicting the cost of an apartment

- Linear Model:

$$a(x) = \mathrm{w}_0 + \mathrm{w}_1 * (\text{area}) + \mathrm{w}_2 * (\text{floor}) + \mathrm{w}_3 * (\text{location}) + \cdots$$

- These features are likely not independent of each other

Possible solution: Feature engineering, i.e. add polynomial features

Example
Predicting the cost of an apartment

- Linear model with polynomial features:

$$
\begin{aligned}
a(x) = \mathrm{w}_0 + \mathrm{w}_1 * (\mathrm{area}) + \ \mathrm{w}_2 * (\mathrm{floor}) + \ \mathrm{w}_3 * (\mathrm{location}) \\
\mathrm{w}_4 * (\mathrm{area})^2 + \ \mathrm{w}_5 * (\mathrm{floor})^2 + \ \mathrm{w}_6 * (\mathrm{location})^2 \\
\mathrm{w}_7 * (\mathrm{area}) * (\mathrm{floor}) \\
+ \cdots
\end{aligned}
$$

- How to interpret this model?

- What is $(\mathrm{area}) * (\mathrm{floor})$?

Example
Predicting the cost of an apartment

Gradient boosting

$$a_N(x) = \sum_{n=1}^{N} b_n(x)$$

The N-th model:

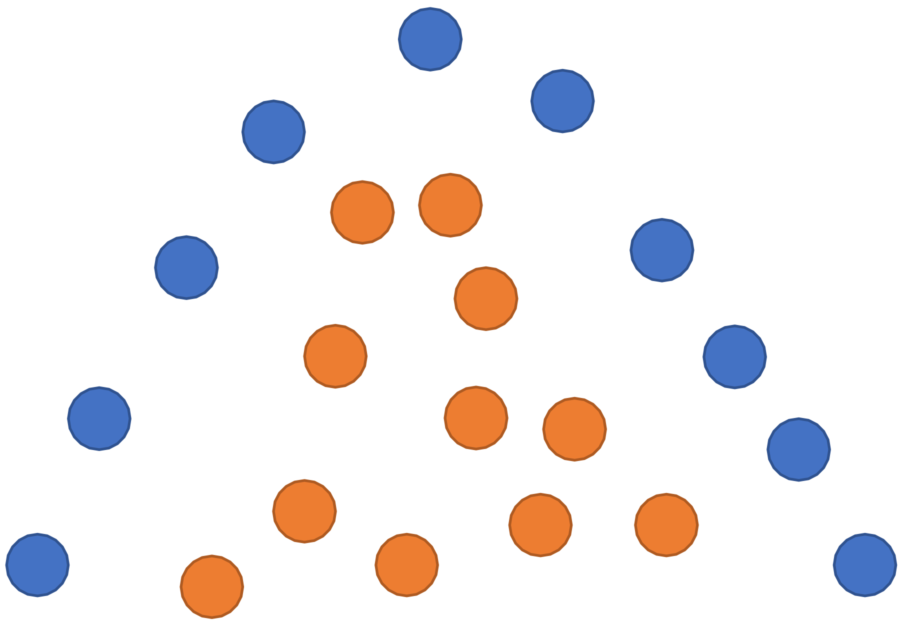
$$a\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b_N(x_i)) \to \max_{b_N(x)}$$

Summary

- Linear models can be trained using gradient descent but are not well-suited for capturing complex patterns
- Decision trees and their ensembles provide better results but are challenging to train

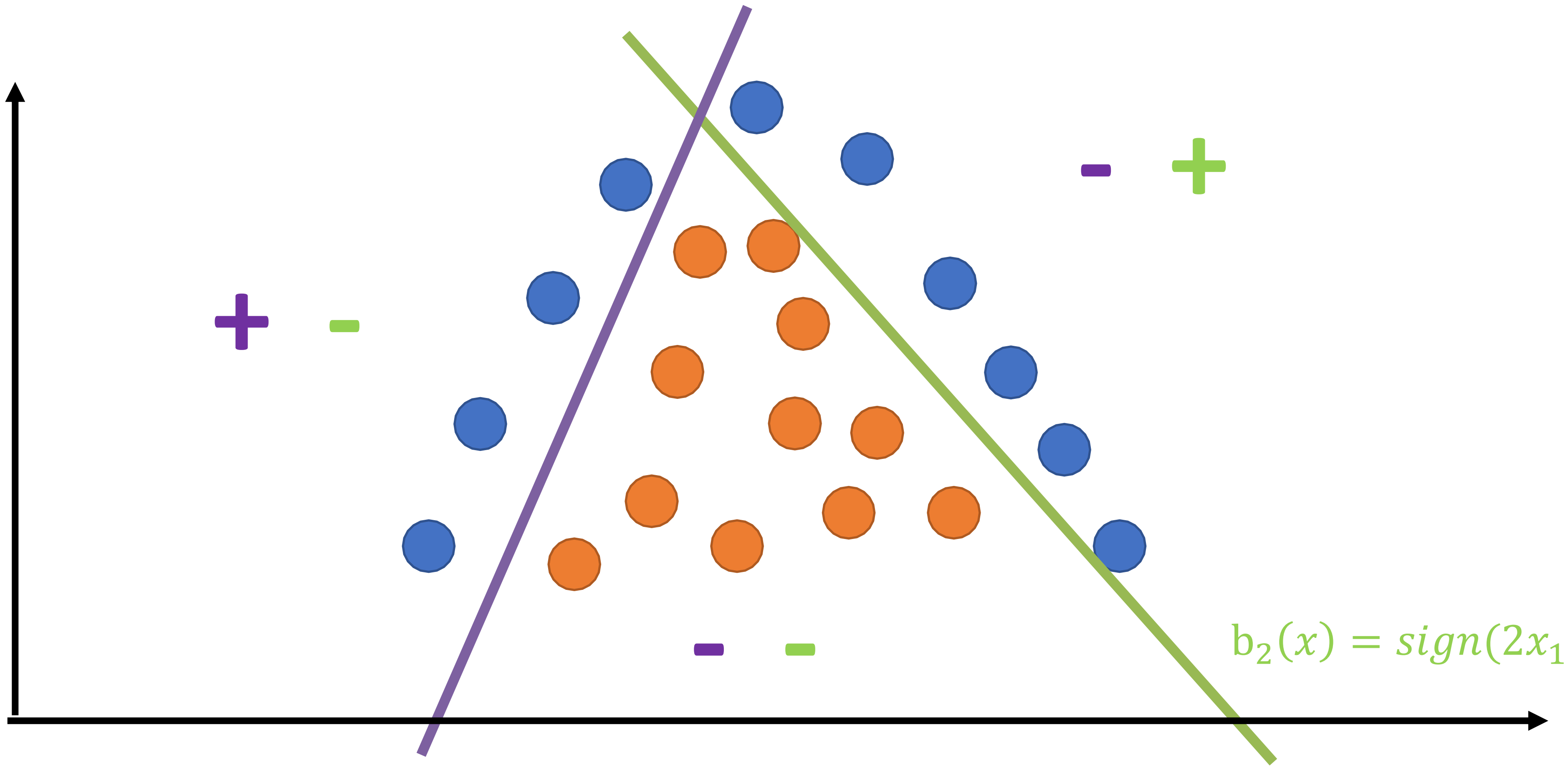
Neural Network can be seen as combination of both linear models and boosting methods!

Neural Network Structure



Nonlinear patterns

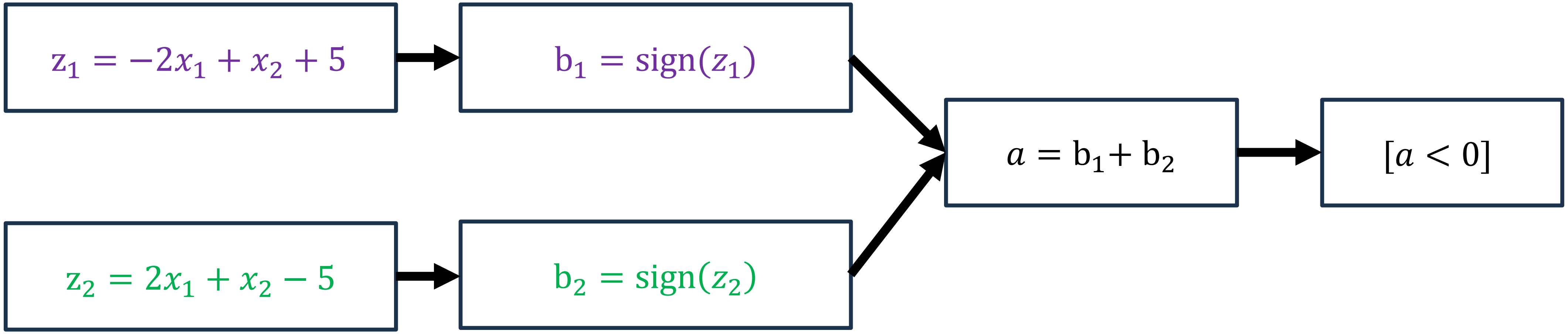
Neural Network Structure



$b_2(x) = sign(2x_1 + x_2 - 5)$

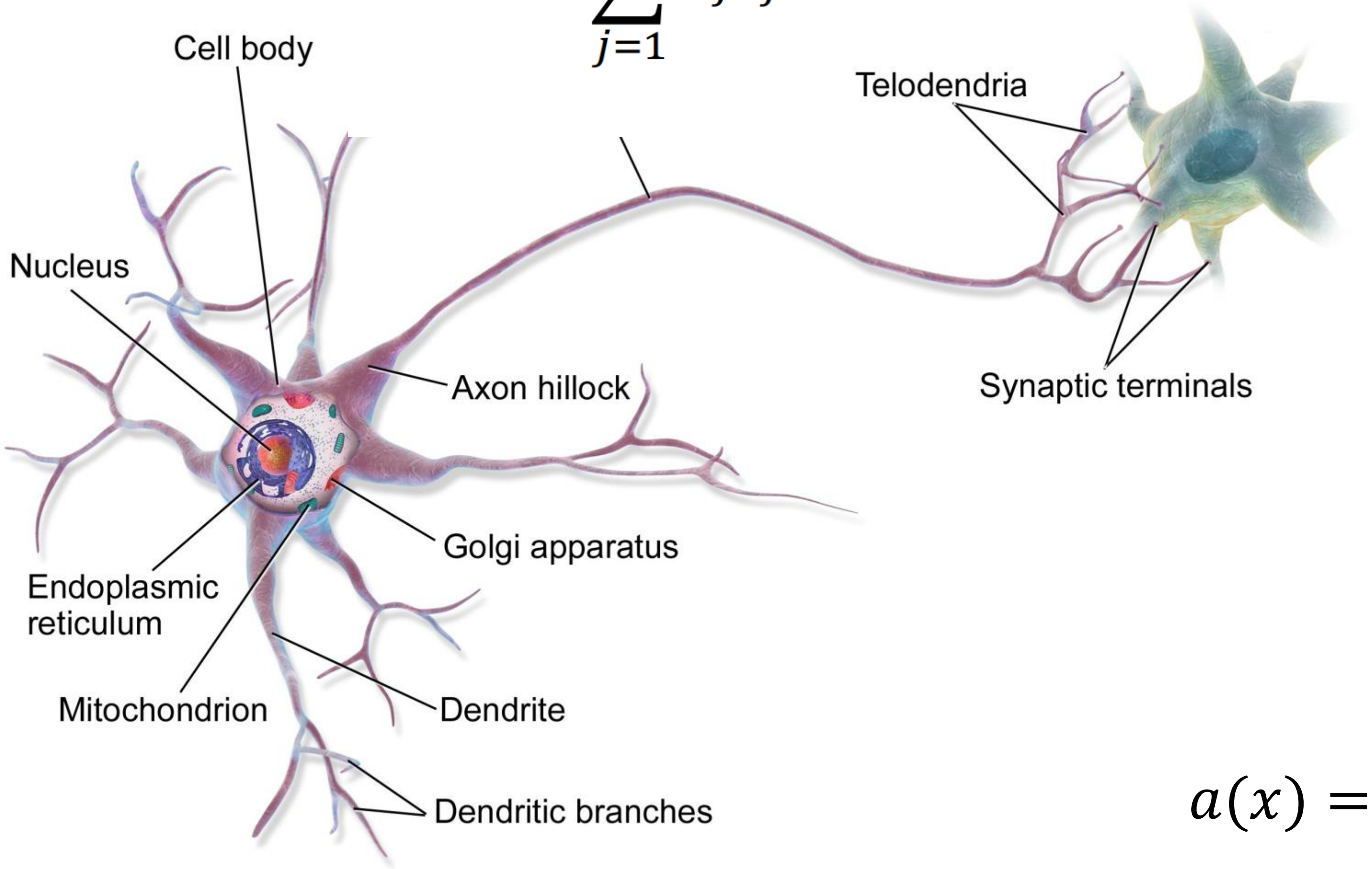
$b_1(x) = sign(-2x_1 + x_2 + 5)$

Neural Network Structure



Neuron

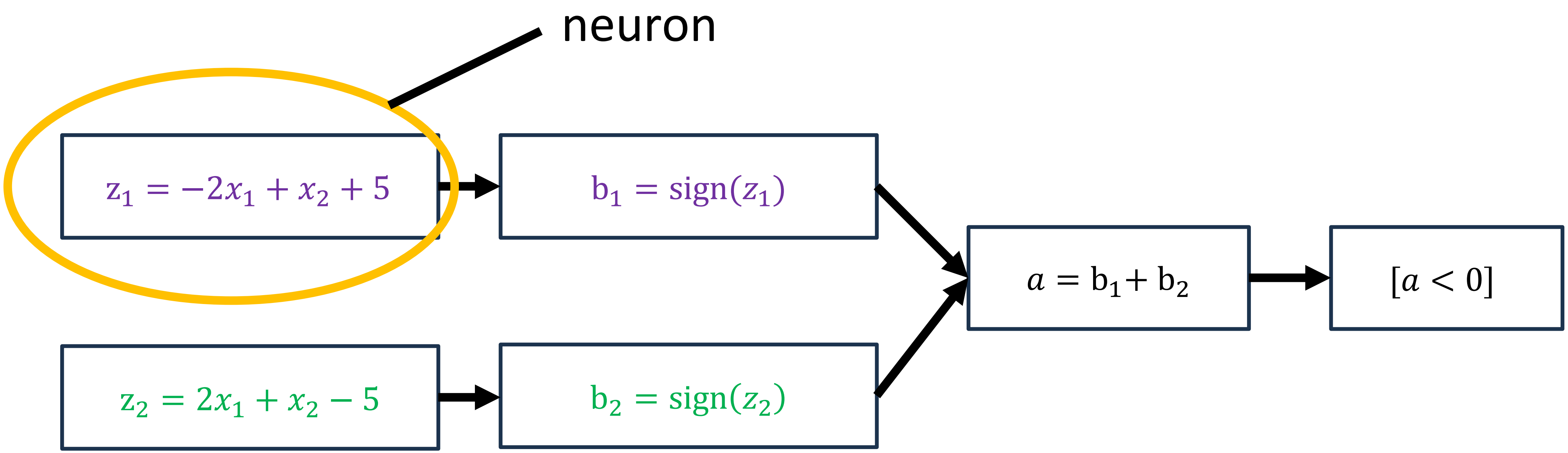
$$a(x) = \sum_{j=1}^{d} w_j x_j$$



$$a(x) = \sum_{i=1}^{d} w_i x_i$$

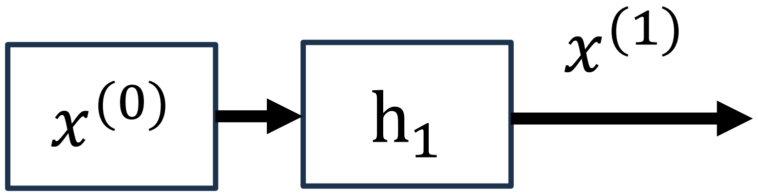
https://en.wikipedia.org/wiki/Neuron

Neural Network Structure

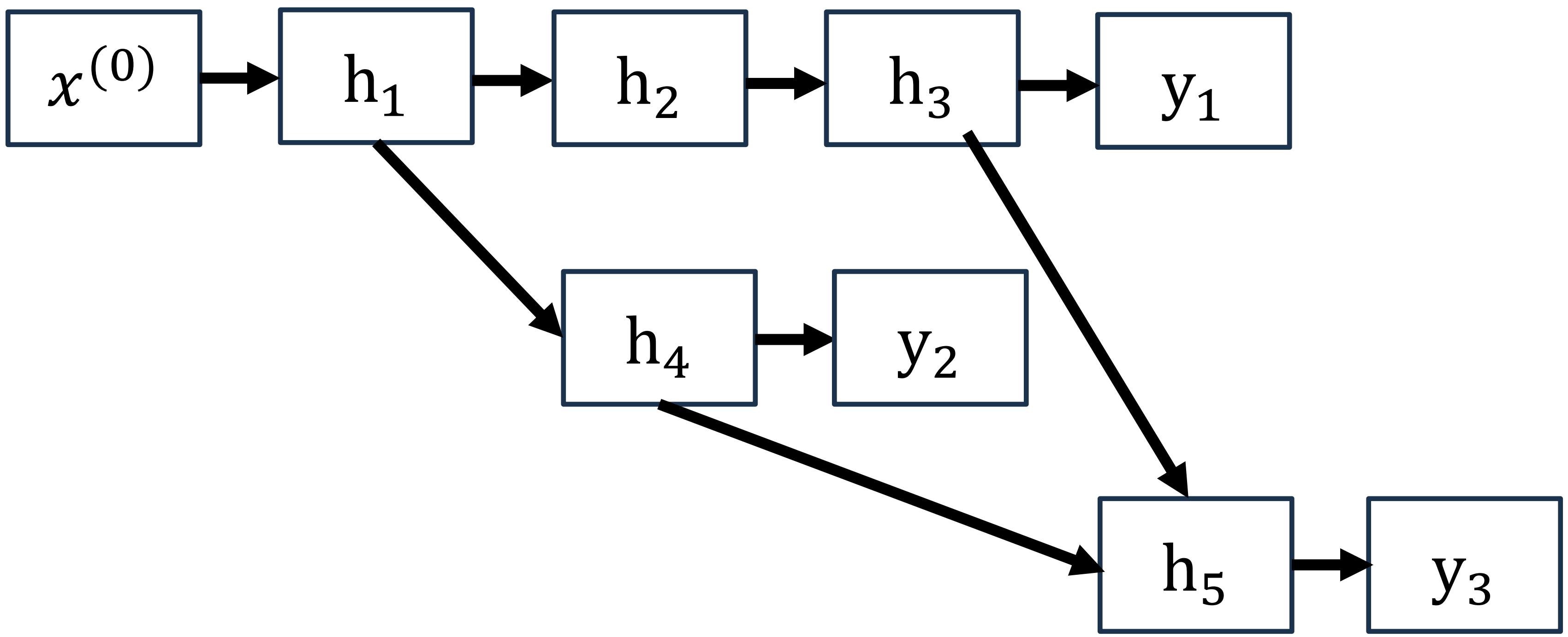


Computational graph (neural network)

- $x^{(0)}$ — input features
- $h_1(x)$ — layer
- $x^{(1)}$ — output



Computational graph (neural network)



Fully connected layers

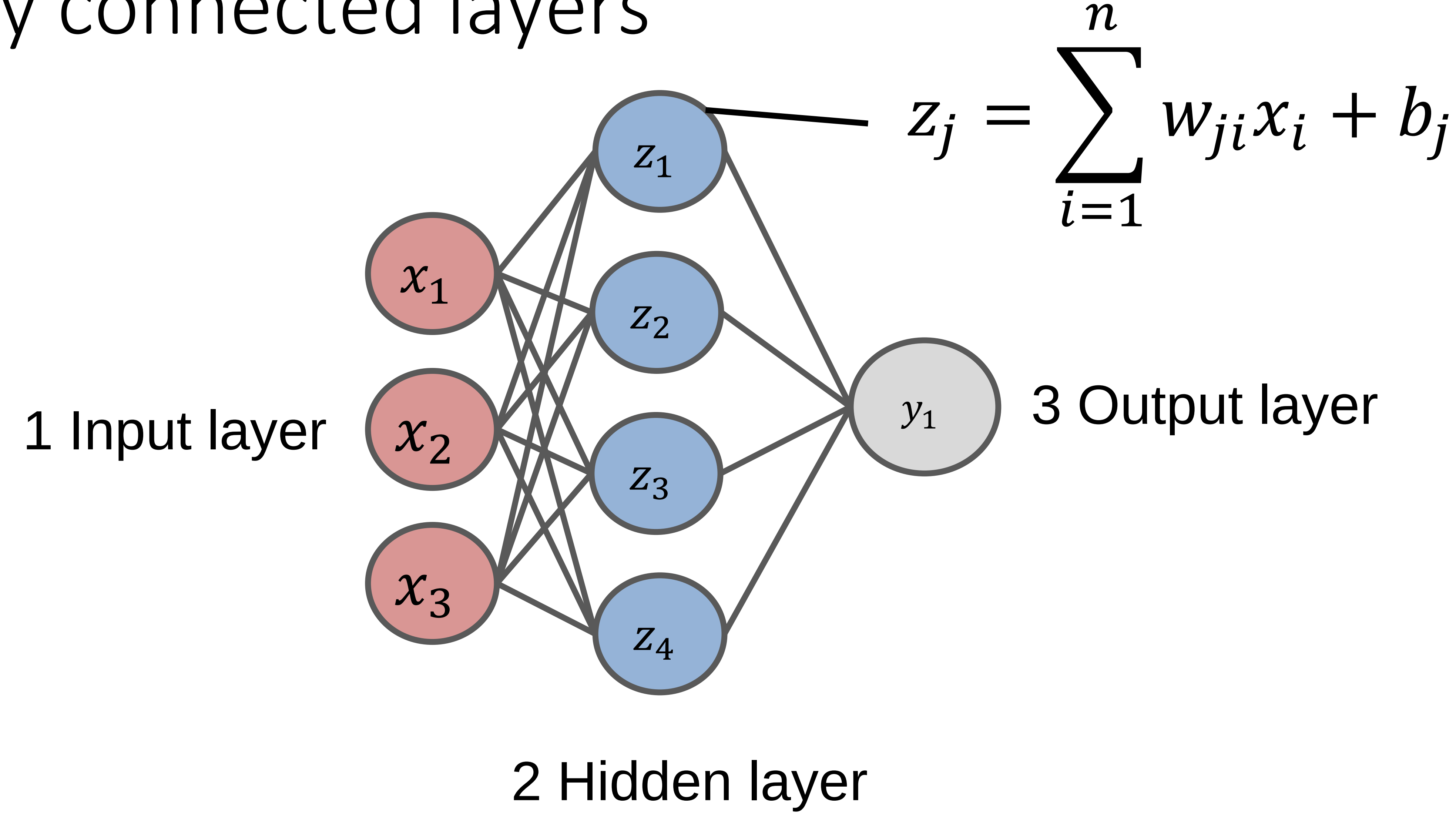
Fully connected layers

- The input consists of N values, and the output produces M values
- $x_1, \cdots x_N$ — input
- $y_1, \cdots y_M$— output

- Each output is the result of applying a linear model to the inputs.

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

Fully connected layers



https://cs231n.github.io/convolutional-networks/

Fully connected layers

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer

torch.nn.Linear(20, 30)

keras.layers.Dense(64)

Fully connected layers

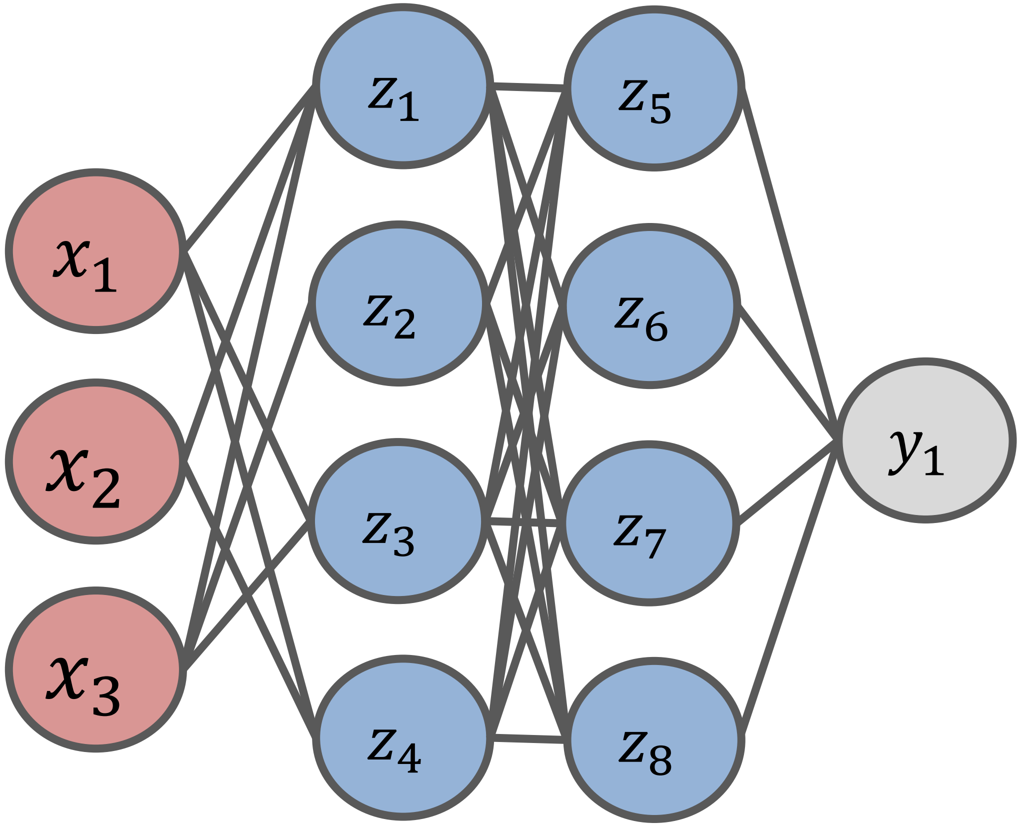
$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer
- if we have 1,000,000 input features and 1000 outputs, it amounts to 1,000,000,000 parameters
- a substantial amount of data is needed for training

Important Questions in DL

How to build a useful model in deep learning?
Which layers to add?

Fully connected layers



- Can we have 2 fully-connected layers one after another?

Non-linearity

- Given 2 fully-connected layers

$$s_k = \sum_{j=1}^{m} v_{kj} z_j + c_k = \sum_{j=1}^{m} v_{kj} \sum_{j=1}^{m} w_{ji} x_i + \sum_{j=1}^{m} v_{kj} b_j + c_k =$$

$$= \sum_{j=1}^{m} (\sum_{j=1}^{m} \boldsymbol{v_{kj} w_{ji}} x_i + \boldsymbol{v_{kj} b_j} + \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{c_k})$$

$$\boxed{z_j = \sum_{i=1}^{n} \boldsymbol{w_{ji}} x_i + \boldsymbol{b_j}}$$

- **So, this is no better than a single fully connected layer**

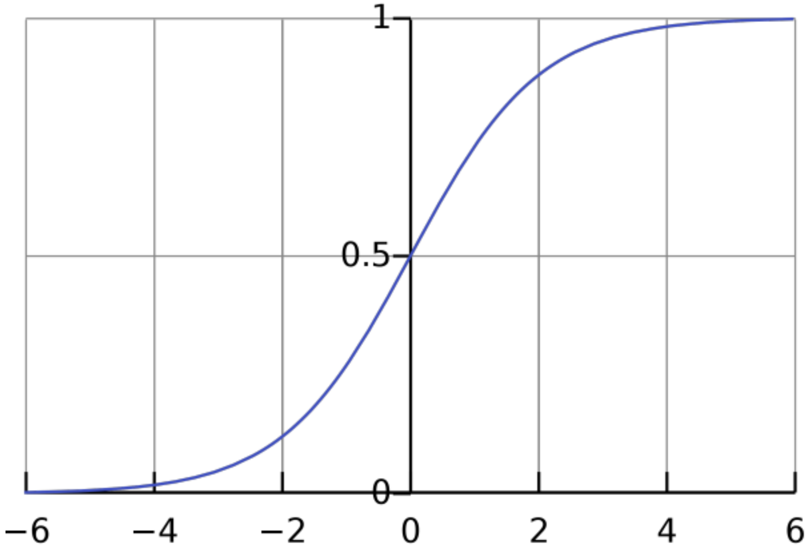
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer (`torch.nn.Sigmoid`)

$$z_j = f(\sum_{i=1}^{n} w_{ji} x_i + b_j)$$

1. $f(x) = \dfrac{1}{1 + \exp(-x)}$

Logistic / Sigmoid



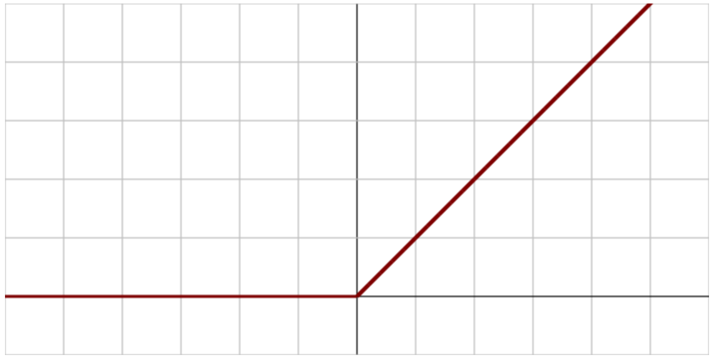
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer (`torch.nn.ReLU`)

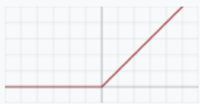
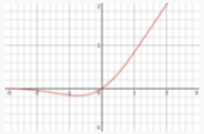
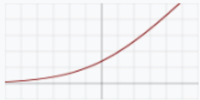
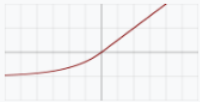
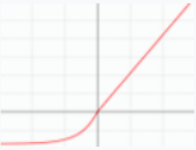
$$z_j = f(\sum_{i=1}^{n} w_{ji} x_i + b_j)$$

2. $f(x) = \max(0, x)$

(ReLU, REctified Linear Unit)

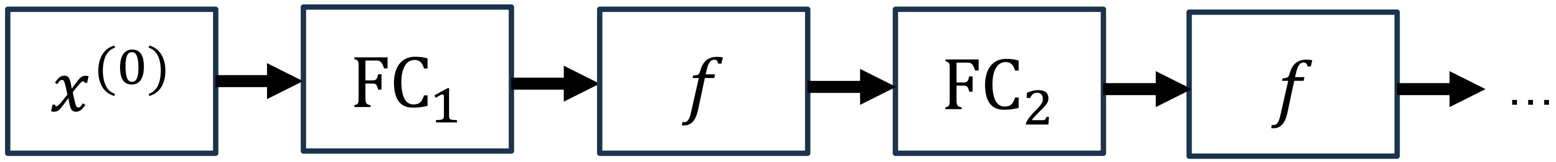


Activation Functions

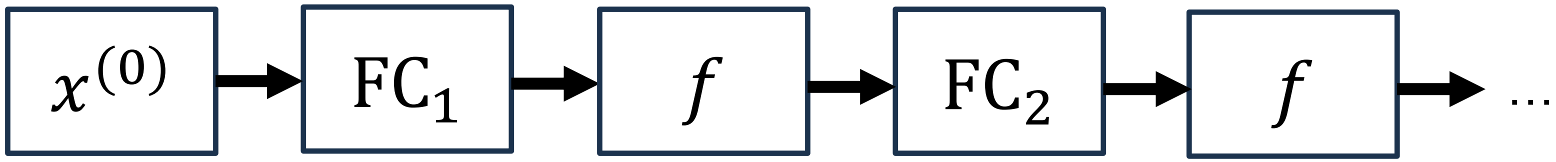
Rectified linear unit (ReLU)[8]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max(0, x) = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[2]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$
Softplus[9]		$\ln(1 + e^x)$
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha\left(e^x - 1\right) & \text{if } x \le 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Scaled exponential linear unit (SELU)[11]		$\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \ge 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$

https://en.wikipedia.org/wiki/Activation_function

A fully connected neural network



A fully connected neural network



- Features are fed into the network
- The dimension of the last layer = # target variables (# classes)

The Cybenko Universal Approximation Theorem

Summary:

Let $g(x)$ be a continuous function. Then, it is possible to construct a two-layer neural network that approximates $g(x)$ with any predefined precision.

In other words, two-layer neural networks are VERY powerful!

Training neural networks

Warm-up

Which of this is the formula for a step in gradient descent?

1. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} + \eta\nabla\mathrm{Q}(\mathrm{w}^{\mathrm{t}})$
2. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} - \eta\nabla\mathrm{Q}(\mathrm{w}^{\mathrm{t}-1})$
3. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} - \eta\nabla\mathrm{Q}(\mathrm{w}^{\mathrm{t}})$
4. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} + \eta\nabla\mathrm{Q}(\mathrm{w}^{0})$

Gradient Descent

- Repeat until convergence

$$w^t = w^{t-1} - \eta \nabla Q(w^t)$$

Updated parameter

Learning Rate

Gradient at the previous point

Convergence

- Stopping rule 1

$$\|w^t - w^{t-1}\| < \varepsilon$$

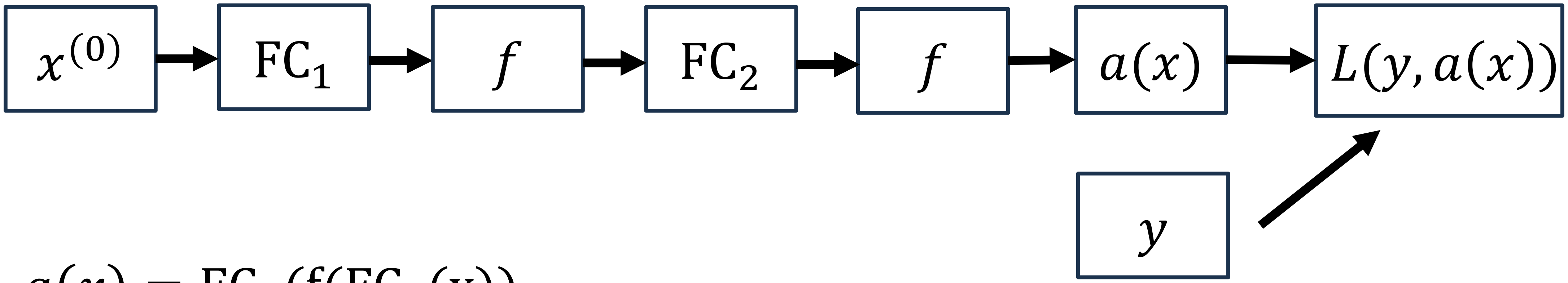
- Stopping rule 2

$$\|\nabla Q(w^t)\| < \varepsilon$$

- In DL: stop when the error on the test set stops decreasing

Training neural networks

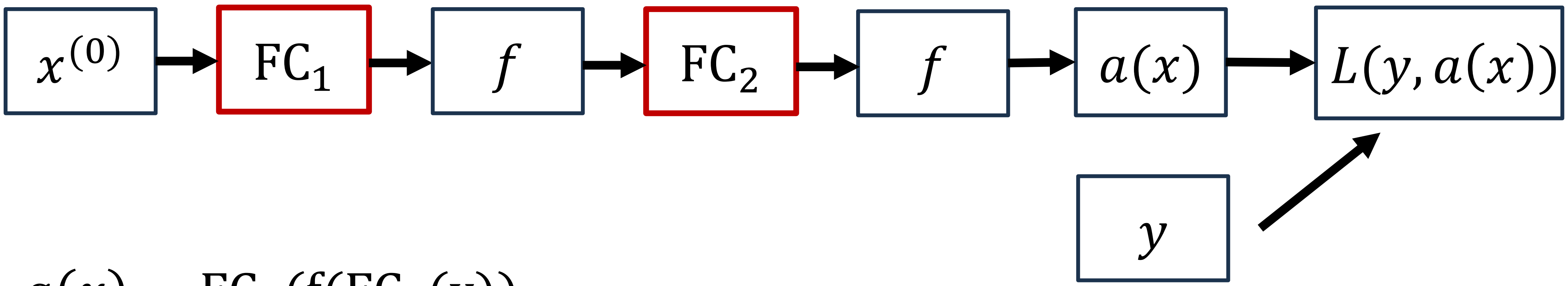
- All layers are usually possible to differentiate, so it's possible to compute derivatives with respect to all parameters



- $a(x) = \mathrm{FC}_2(\mathrm{f}(\mathrm{FC}_1(\mathrm{x}))$
- Where are the parameters in this neural network?

Training neural networks

- All layers are usually possible to differentiate, so it's possible to compute derivatives with respect to all parameters



- $a(x) = \mathrm{FC}_2(\mathrm{f}(\mathrm{FC}_1(\mathrm{x})))$
- Where are the parameters?

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, a(x_i)) \to \min_a$$

Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2$$

$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2 = 2(a(x_i, w) - y_i)\frac{\partial}{\partial w_j}a(x_i, w)$$

Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

$$\frac{\partial}{\partial w_j}(a(x_i,w)-y_i)^2 = 2(a(x_i,w)-y_i)\frac{\partial}{\partial w_j}a(x_i,w)$$

- $a(x_i,w)=10, y_i=9.99$: $\quad 2*0.01*\frac{\partial}{\partial w_j}a(x_i,w)$
- $a(x_i,w)=10, y_i=1$: $\quad 2*9*\frac{\partial}{\partial w_j}a(x_i,w)$

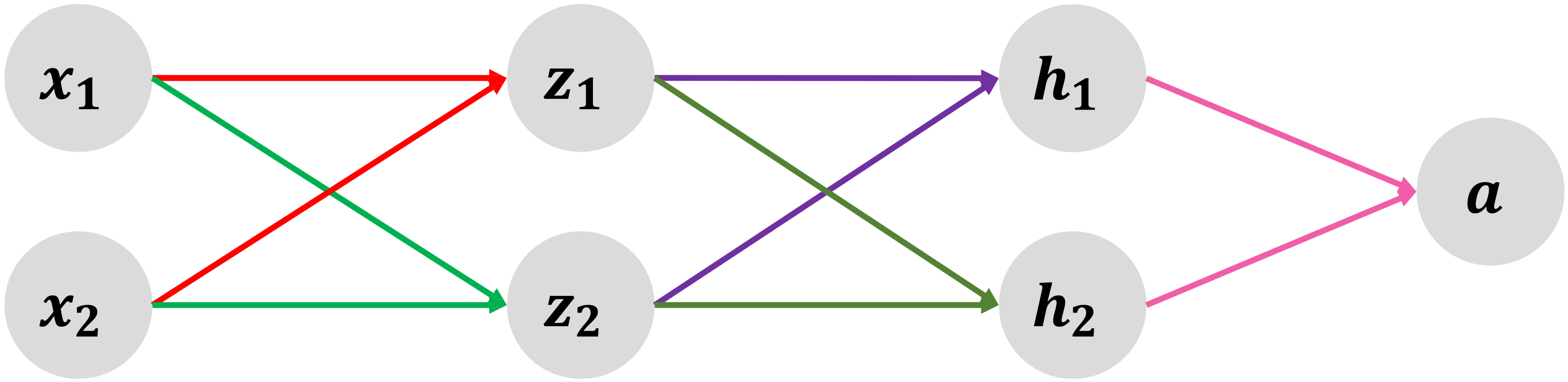
Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

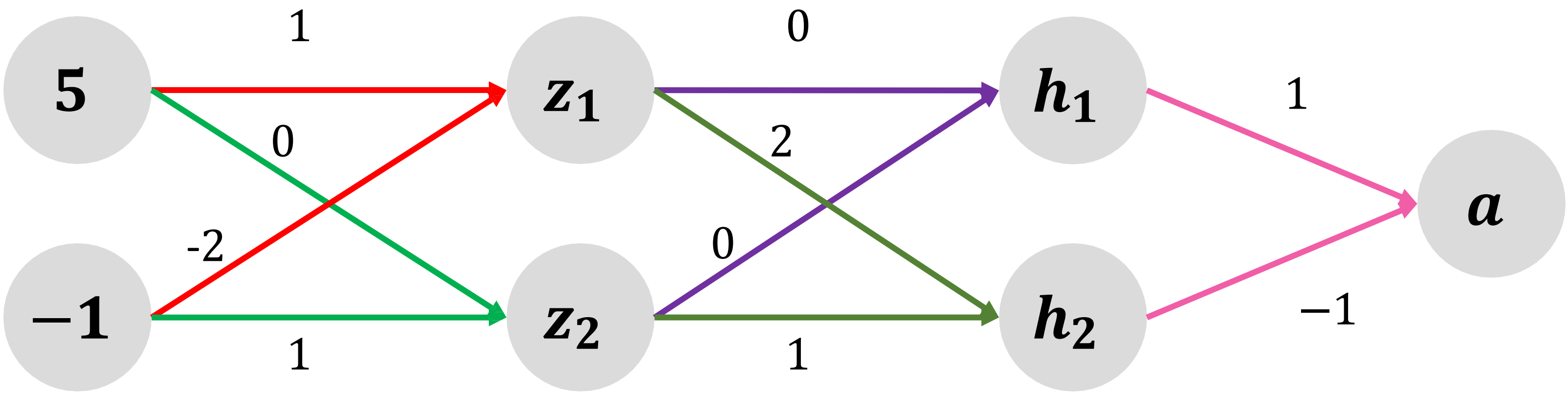
$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2 = 2(a(x_i, w) - y_i)\frac{\partial}{\partial w_j}a(x_i, w)$$

$$\frac{\partial}{\partial w_j}L(y_i, a(x_i, w)) = \frac{\partial}{\partial z}L(y_i, z)\Big|_{z=a(x_i, w)} \frac{\partial}{\partial w_j}a(x_i, w)$$

How to compute derivatives?

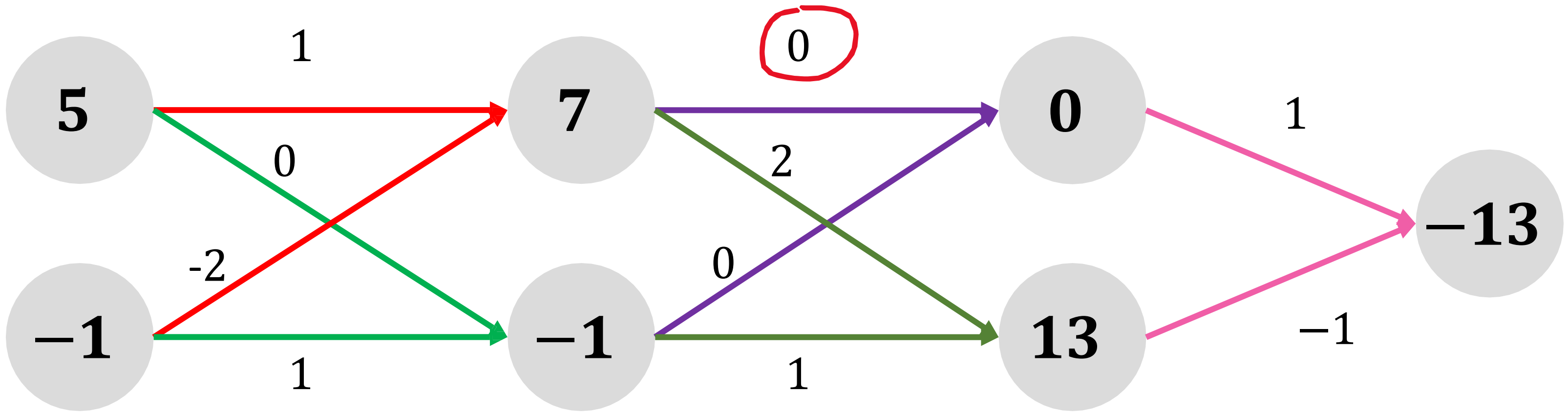


How to compute derivatives?

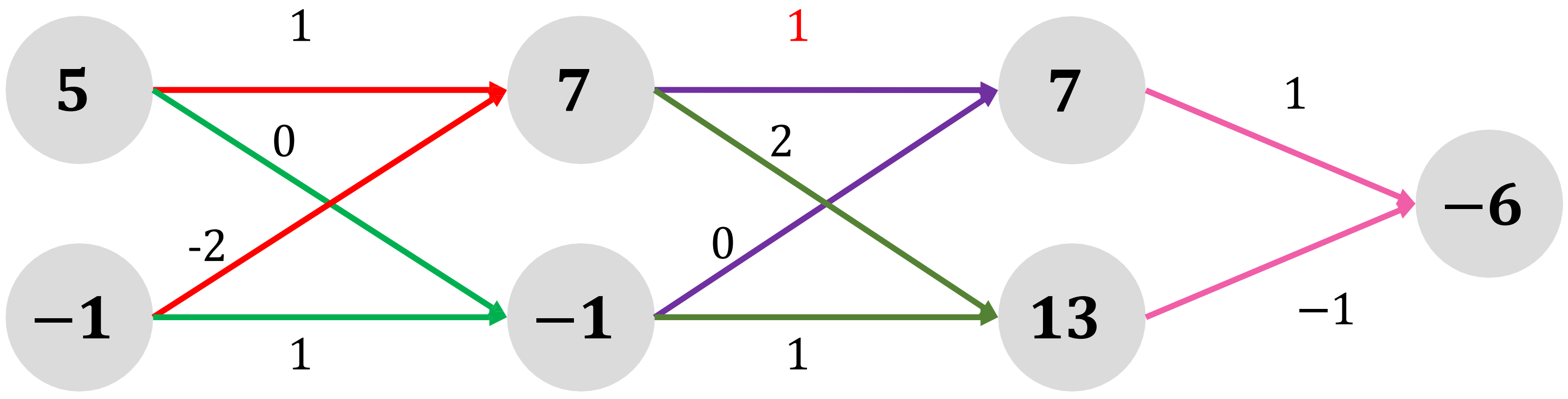


How to compute derivatives?

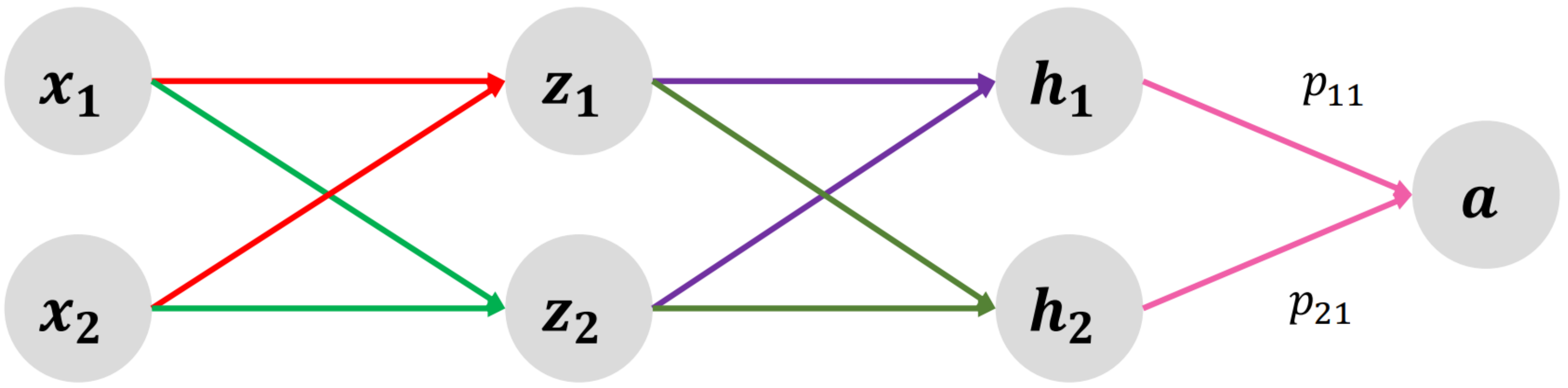
Will the output (-13) change if we change this weight from 0 to 1?



How to compute derivatives?



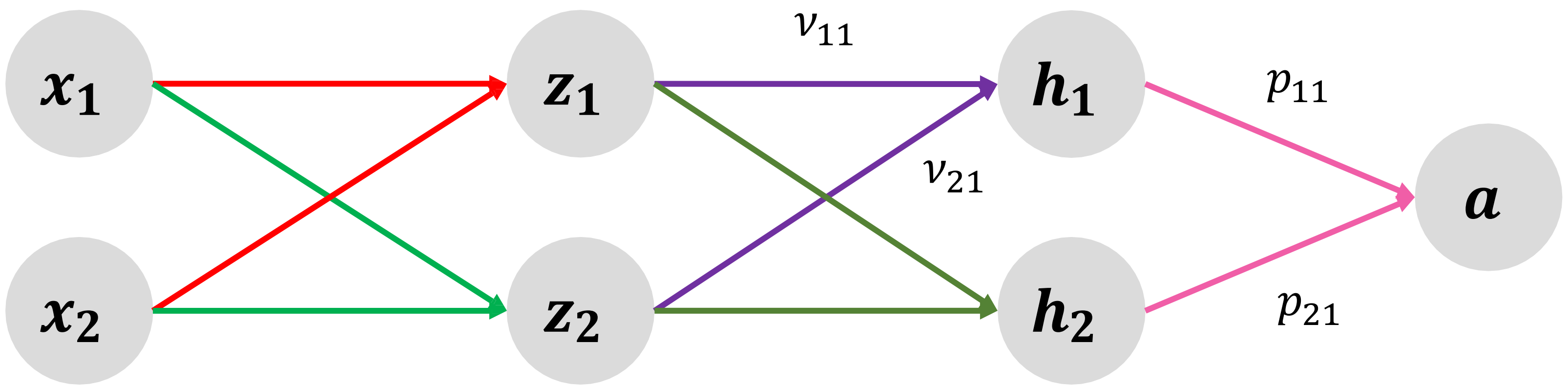
How to compute derivatives?



$$a(x) = p_{11}h_1(x) + p_{21}h_2(x)$$

$$\frac{\partial a}{\partial p_{11}} = h_1(x)$$

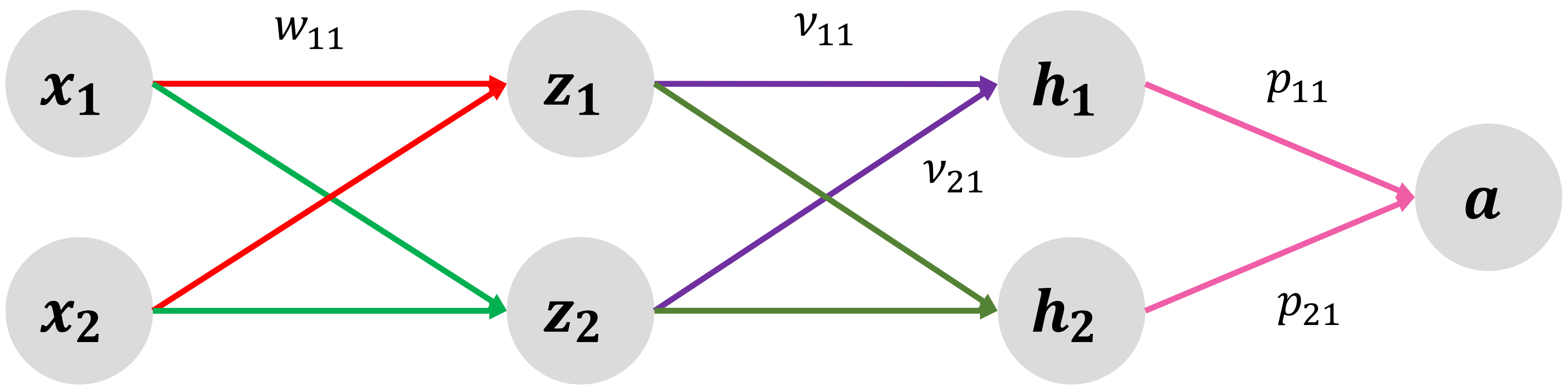
How to compute derivatives?



$$a(x) = p_{11} f(v_{11} z_1(x) + v_{21} z_2(x) +) + p_{21} h_2(x)$$

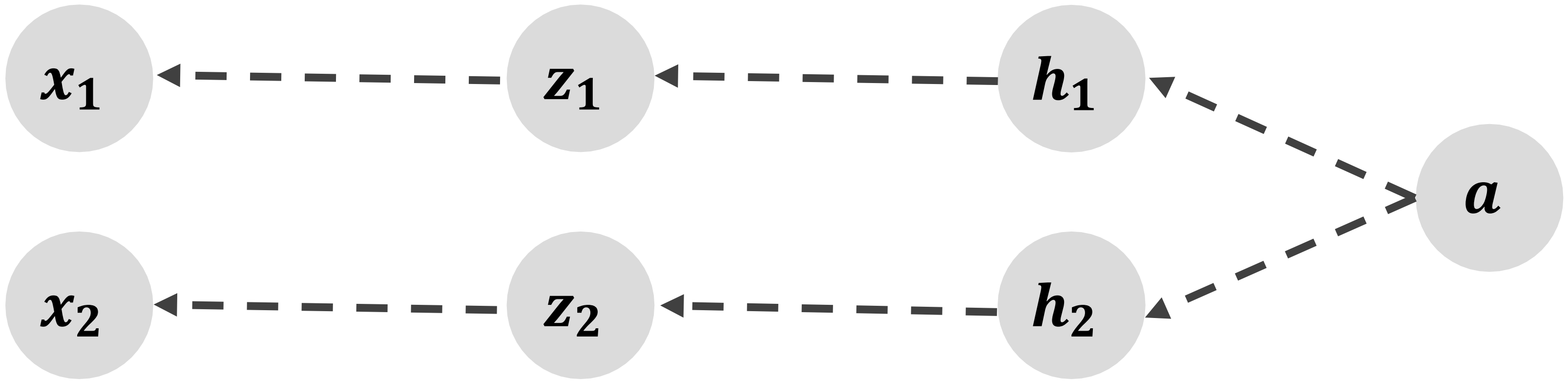
$$\frac{\partial a}{\partial v_{11}} = \frac{\partial a}{\partial h_1} \frac{\partial h_1}{\partial v_{11}}$$

How to compute derivatives?



$$\frac{\partial a}{\partial w_{11}} = \frac{\partial a}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial w_{11}} + \frac{\partial a}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial w_{11}}$$

How to compute derivatives?



- We move in the opposite direction along the graph and calculate derivatives
- Backpropagation

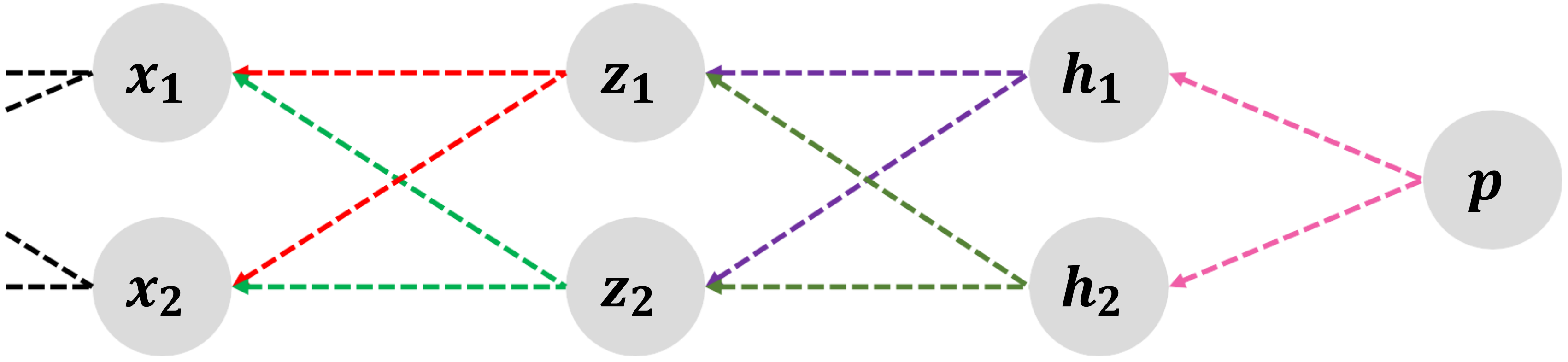
3: $\dfrac{\partial p}{\partial h_1}$ $\dfrac{\partial p}{\partial h_2}$

2: $\dfrac{\partial p}{\partial z_1} = \dfrac{\partial p}{\partial h_1}\dfrac{\partial h_1}{\partial z_1} + \dfrac{\partial p}{\partial h_2}\dfrac{\partial h_2}{\partial z_1}$ $\dfrac{\partial p}{\partial z_2} = \dfrac{\partial p}{\partial h_1}\dfrac{\partial h_1}{\partial z_2} + \dfrac{\partial p}{\partial h_2}\dfrac{\partial h_2}{\partial z_2}$

1:

$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_2}$$



Backpropagation

- Many formulas have the same derivatives
- In backpropagation, each partial derivative is computed only once—computing derivatives for layer N is reduced to multiplying the matrix of derivatives for layer N+1 by some vector

Fully connected networks for image classification

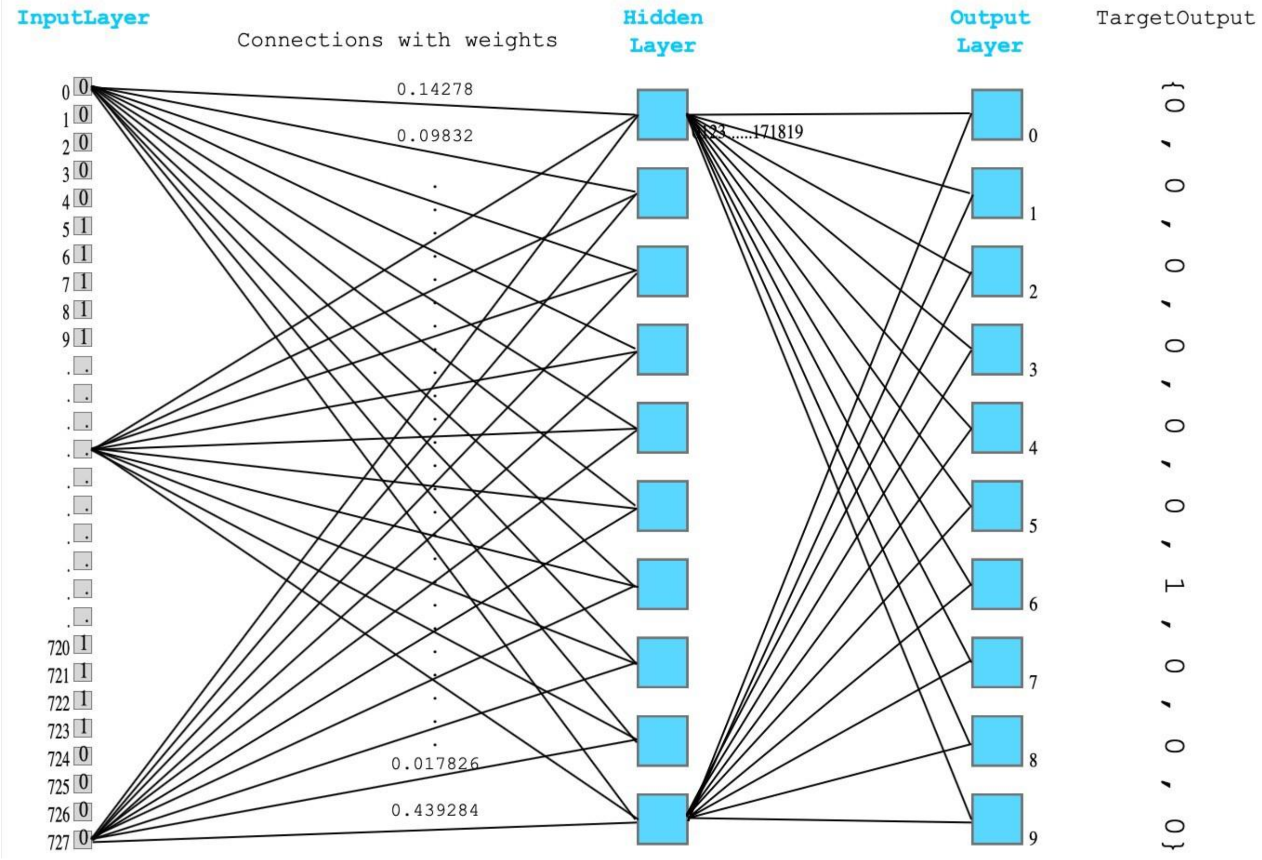
MNIST

MNIST

- Images sizes are 28 x 28
- Images are centered
- 60,000 objects in the training set

MNIST

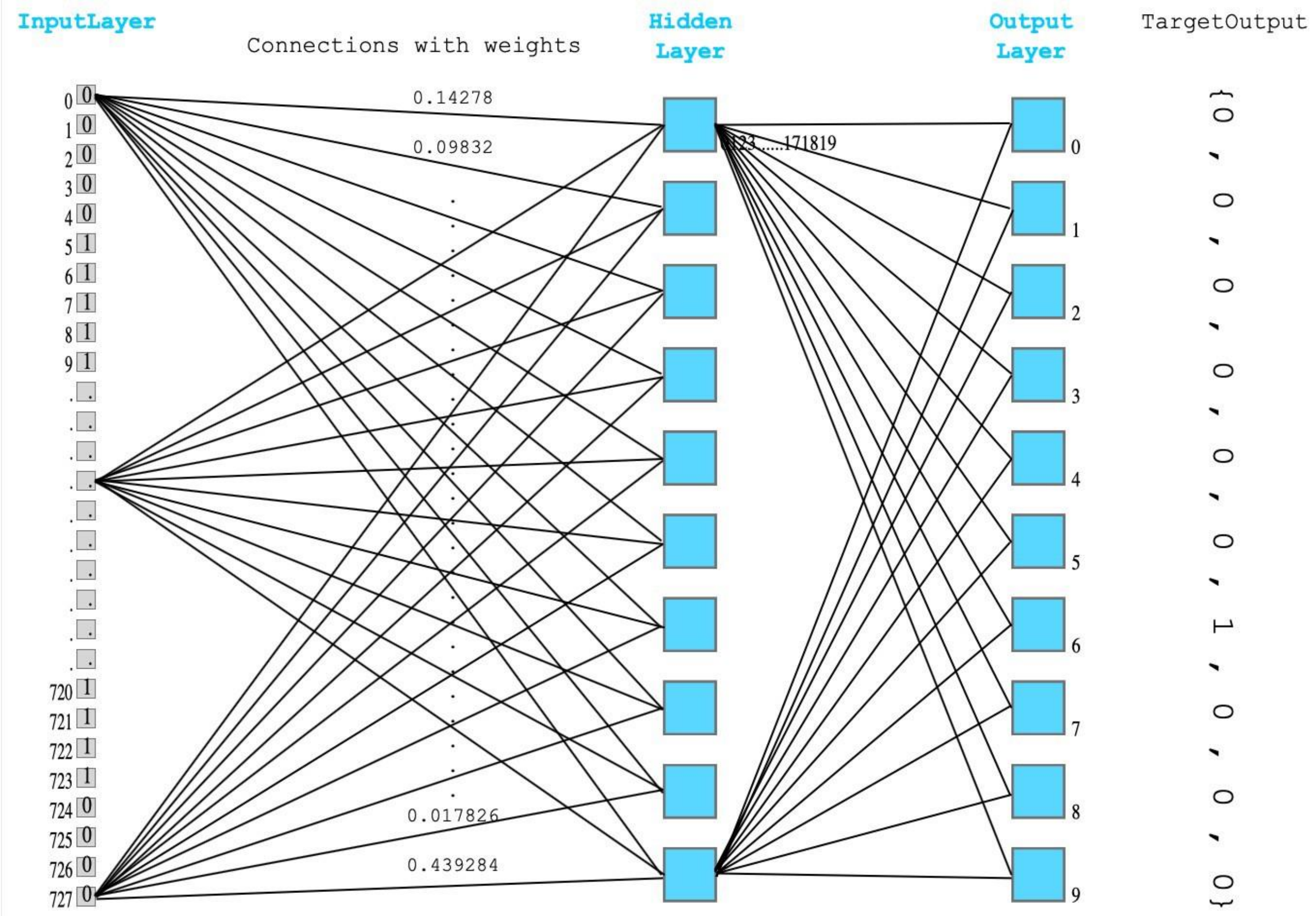
- What will a fully connected network learn?



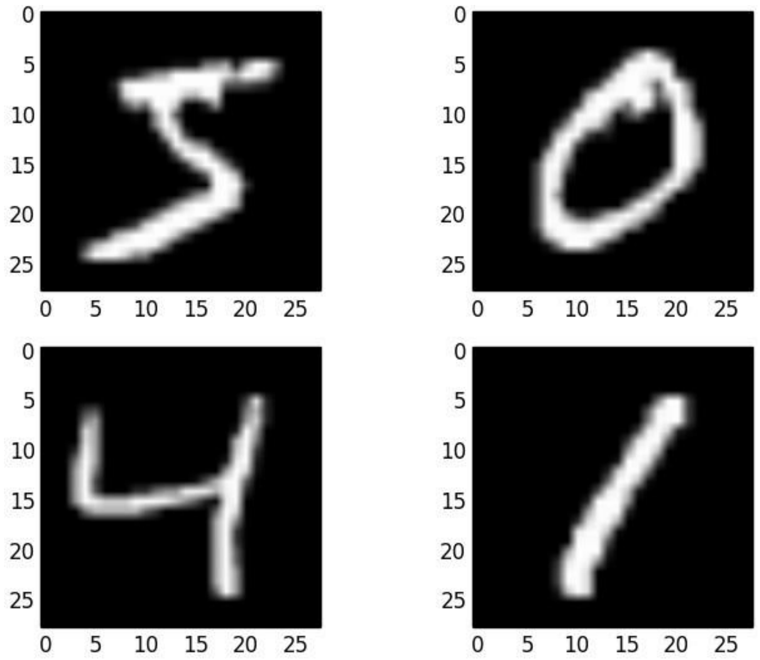
https://mmlind.github.io/Simple_3-Layer_Neural_Network_for_MNIST_Handwriting_Recognition/

MNIST

- Each neuron can detect the presence of a specific set of pixels

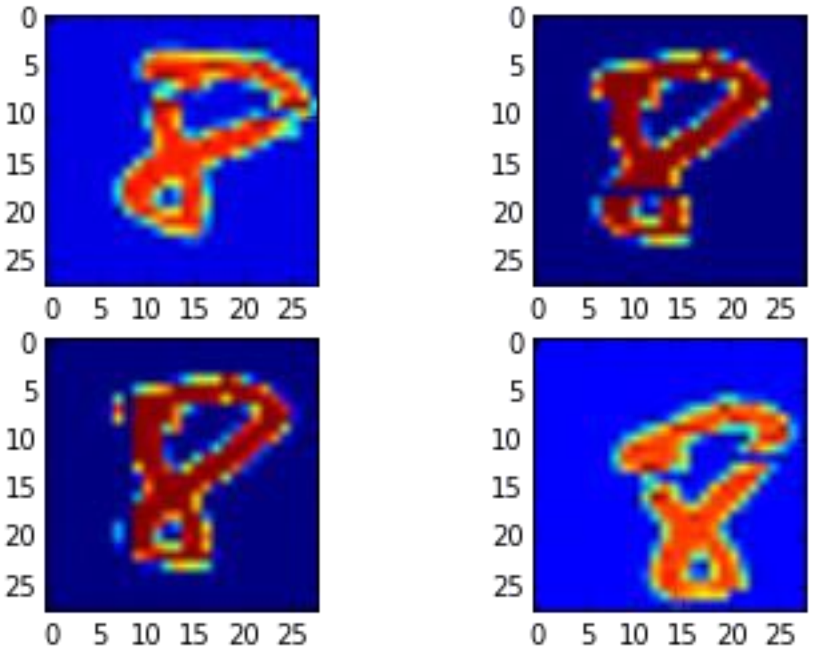


MNIST



MNIST

- If you shift the digit slightly, the neuron will no longer detect its pattern



https://srome.github.io/Jitter,-Convolutional-Neural-Networks,-and-a-Kaggle-Framework/

Number of parameters

- 784 inputs
- Fully connected layer: 1000 neurons
- Output layer: 10 neurons (one for each class)
- Weights between input and fully connected layers:
  (784 + 1) * 1000 = 785,000
- Weights between fully connected and output layers:
  (1000 + 1) * 10 = 10,010

Fully connected neural networks for image classification

- A lot of parameters
- Prone to overfitting
- Does not consider the specifics of images (shifts, slight changes in shape, etc.)
- One of the best ways to combat overfitting is to reduce the number of parameters